REPORT DO

AD-A236 483

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estin
maintaining the data needed, and completing and reviewing the c
suggestions for reducing this burden, to Washington Headquarte
and to the Office of Management and Budget, Paperwork Reduc... Project (0704-0188), Washington, DC 20503

ctions, searching existing data sources, gathering and
y other aspect of this collection of information, including
Davis Highway, Suite 1204, Arlington, VA 22202-4302

1 AGENCY USE ONLY *(Leave blank)*	2 REPORT DATE	3 REPORT TYPE AND DATES COVERED
	May 1991	Professional Paper

4 TITLE AND SUBTITLE

IMPLEMENTATION OF VQ ALGORITHMS ON A RECONFIGURABLE ARRAY PROCESSOR

5 FUNDING NUMBERS

PR: EE33
WU: DN 308026
PE: 0603218C

6 AUTHOR(S)

T. B. Henderson and K. S. Thyagarajan

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152–5000

8 PERFORMING ORGANIZATION REPORT NUMBER

9 SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

10 SPONSORING/MONITORING AGENCY REPORT NUMBER

11 SUPPLEMENTARY NOTES

DTIC COPY INSPECTED 5

12a DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b DISTRIBUTION CODE

A-1

13 ABSTRACT *(Maximum 200 words)*

Vector quantization is being widely used in image data compression applications due to the facts that it is capable of achieving fractional bit rates with reasonable complexity and that the decoding is a very simple table look-up scheme. In image encoding, a vector quantizer accepts a block of pixels and outputs an address of the best matching tile stored in a codebook. The matching algorithm requires a large number of basic arithmetic operations in typical applications. Since real-time coding is required in many video applications, the need for dedicated processing architectures arises naturally. This paper investigates the mapping of VQ algorithms onto an array processor to achieve near real-time compression of video images.

91-01221

Published in *Proceedings 24th Annual Asilomar Conference on Signals, Systems and Computers*, Nov 1990.

14 SUBJECT TERMS	15 NUMBER OF PAGES
parallel processing image processing parallel computers	16 PRICE CODE

17 SECURITY CLASSIFICATION OF REPORT	18 SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20 LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

IMPLEMENTATION OF VQ ALGORITHMS ON A RECONFIGURABLE ARRAY PROCESSOR

Thomas B. Henderson

Naval Ocean Systems Center
Code 761
San Diego, CA 92152

K. S. Thyagarajan

San Diego State University
Electrical & Computer Engineering Department
San Diego, CA 92182

ABSTRACT

Vector quantization is being widely used in image data compression applications due to the facts that it is capable of achieving fractional bit rates with reasonable complexity and that the decoding is a very simple table look-up scheme. In image encoding, a vector quantizer accepts a block of pixels and outputs an address of the best matching tile stored in a codebook. The matching algorithm requires a large number of basic arithmetic operations in typical applications. Since real-time coding is required in many video applications, the need for dedicated processing architectures arises naturally. This paper investigates the mapping of VQ algorithms onto an array processor to achieve near real-time compression of video images.

1. INTRODUCTION

Image data compression algorithms are used to reduce the number of bytes required to represent a digitally encoded image. The primary applications of image compression are to minimize communication bandwidth for image transmission and to minimize the amount of memory required for image storage. Typical television images have about 512x512 pixels per frame with a frame rate of 30 frames/s. When digitally encoded at 8-bits per pixel intensity resolution, the required digital transmission rate is nearly 60 million bits/s [1]. High quality color graphics displays have 1024x1024 24-bit pixels per frame requiring 3 MBytes of storage space. As frame sizes increase, the need for efficient image compression algorithms becomes even more acute.

Image compression algorithms are characterized by compression rate, distortion, and computational complexity. The compression ratio is determined by dividing the number of bytes required to represent an image by the number of bytes needed to represent the compressed image. Distortion is a measure of the error introduced into an image by the encoding and decoding processes. Computational complexity is an indication of the number of arithmetic calculations necessary to compress and decompress an image. Computational complexity is a very important consideration in algorithm implementation, especially for image transmission applications where the time available for encoding is limited by the frame rate.

Most compression algorithms can be classified either as scalar quantizers or vector quantizers. Scalar quantization algorithms achieve compression by encoding each input data value into a single codeword while vector quantizers compress data by mapping a sequence or group of scalar values into a single codeword [2]. A result of Shannon's rate-distortion theory is that better performance is always achievable "in theory" by coding vectors instead of scalars. This makes vector quantization algorithms attractive for applications requiring high compression rates.

A disadvantage of vector quantizers is that they are often more complex than scalar quantizers. For example, the tree-search vector quantization algorithm implemented in this work would require about 400 million integer arithmetic operations per second to encode the television image described above. Currently, single DSP or microprocessor chips (such as the AT&T DSP32C or Intel i860) cannot maintain this computational rate [3, 4]. Array processors provide a feasible solution to this problem and can be scaled to meet the computational requirements of different applications.

The Video Analysis Transputer Array (VATA) is a flexible reconfigurable array processor that has been designed and built at Naval Ocean Systems Center (NOSC) in San Diego [5, 6]. The VATA is an array of Inmos T800 transputers connected with software reconfigurable communication links. It has been designed for use as a variable architecture array processor testbed for matching optimal array configurations to a wide variety of image and signal processing algorithms. The array resides in an IBM PC-AT host and has a high speed interface to a frame grabber for image processing applications.

Two vector quantization (VQ) algorithms have been mapped onto the VATA. Optimal array architectures have been found for each quantizer. The effects on performance of code optimizations, memory limitations in processing nodes, and overlapping of communication and computation have also been investigated.

2. VECTOR QUANTIZATION ALGORITHMS

The full and binary tree-search VQ algorithms are considered in this work for implementation on a reconfigurable array [2]. The main advantage of the full search algorithm is a moderate memory requirement. Unfortunately, full search is extremely computationally intensive. The binary tree-search algorithm needs twice as much memory but can encode an image with far fewer arithmetic operations.

2.1 Full Search VQ Algorithm

The full search algorithm encodes an input vector X into an index (or "codeword") i using a fixed set of vectors Y_m known as a "codebook". The input vector is compared to each of the vectors in the codebook. A distortion measure D_m is calculated for each Y_m codebook vector and the minimum distortion D_i is found corresponding to codebook entry Y_i. The index i of the codebook vector that produced the minimum distortion is the output of the encoder. The decoder operates in reverse, taking the index i as input and producing vector Y_i from an identical codebook as output. A block diagram of this algorithm is shown in Figure 1. The compression ratio is calculated by dividing the number of bytes in each input vector by the number of bytes required to represent the index.

For image compression, vectors are formed from p×p blocks of pixels taken from the image to be encoded. Each p×p block is mapped into a $p^2 \times 1$ vector for input to the encoder and each $p^2 \times 1$ decoded output vector is reorganized into a p×p block in the output image. The distortion measure is the square of the Euclidean distance between vectors X and Y

$$d(X, Y) = \sum_{i=0}^{(p^2-1)} (x_i - y_i)^2$$

where d is the distortion and x_i and y_i are the elements of vectors X and Y respectively.

A full search VQ with an R-bit index requires 2^R distortion measure calculations for each input vector and a codebook size of 2^R vectors. Each distortion calculation requires p^2 * (2 adds + 1 multiply) integer arithmetic operations. Each vector stored in the codebook will occupy p^2 bytes at 8-bits per pixel.

2.2 Binary Tree-Search VQ Algorithm

The codebook of a binary tree-searched VQ with an R-bit index is organized in R levels. Each level, L_i, contains 2^i entries grouped in pairs, where i = 1, 2,...,R. An input vector is compared with the two entries of the first codebook level and the "closest" codebook vector is selected using the minimum distortion measure. The result determines which pair of codebook vectors in the second level will be compared to the input vector. The process is repeated for each level of the codebook with two distortion measure calculations being made at each level. The binary decisions made during the "path" through the codebook "tree" form the output codeword.

A tree-searched VQ with an R-bit index requires only 2 * R distortion measure calculations for each input vector making this algorithm much more attractive for transmission applications. Since an exhaustive search of the codebook is not performed, decoded image quality is slightly degraded compared to the full search algorithm. A second disadvantage of tree-searched VQ is that 2 * (2^R - 1) vectors must be stored in the codebook nearly doubling the memory requirements of full search VQ.

3. THE VIDEO ANALYSIS TRANSPUTER ARRAY

The VATA uses one Inmos 20MHz T800 transputer at each array node. The T800 includes a 32-bit central processing unit (CPU), a 64-bit floating-point unit (FPU), 4 serial communication links, 4 Kbytes of on-chip RAM, and an external memory interface on a single chip [7]. The CPU can achieve a sustained performance of 10 million instructions per second (MIPS). Each serial link can transfer data between memory and another link at 2.35 Mbytes/sec (bidirectional). After initialization, data transfer on one or more links can occur simultaneously with CPU and FPU operations. This important feature allows the overlapping of communication and processing with very little performance degradation. An additional transputer device, the C004 crossbar switch, provides programmable configuration of the array architecture.

A block diagram of the VATA hardware is shown in Figure 2. The system consists of a standard NTSC camera, an RGB display, and an IBM PC-AT host housing two commercially-available boards (the Frame Grabber and the Transputer Add-In Board) and two types of custom boards (the VATA Interface and the VATA Processor).

The frame grabber is a Data Translation model DT2861 Arithmetic Frame Grabber for the IBM PC-AT. The frame grabber can acquire video frames from the camera, store, and display them on the monitor at a frame rate of 30 frames/sec. The frame grabber also has a high speed I/O port which can transfer frame data to or from an external device at 10 Mbytes/sec.

Each VATA Processor (VP) board contains thirty-two T800 transputers and four C004 crossbar switches. Due to board size limitations, VP T800s have no external RAM. The VATA Interface (VI) board handles communication between the frame grabber I/O port, the IBM PC-AT host, and one or more VP boards. Frame data is passed between the frame grabber I/O port and the VP boards while control and status messages are passed between the IBM PC-AT bus and the VP boards. The Transputer Add-In board (Inmos model IMS B008) is used to compile, link, configure, and load programs onto the VI and VP boards using the Inmos Occam Toolset software.

4. VATA SOFTWARE DEVELOPMENT

To map an algorithm onto the VATA, software must be developed for the IBM PC-AT host and for the transputers. The host program configures the array, controls the frame grabber display and acquisition functions, and synchronizes data transfer between the transputers and the frame grabber. The IBM PC-AT host is programmed in C using the Microsoft C compiler, version 5.0.

The development of transputer programs involves the selection of an array architecture optimized for the algorithm and the division of the algorithm into sub-tasks. Each transputer is assigned a sub-task and each sub-task requires a different program. Occam, a parallel programming language designed by Inmos, is currently the optimum language for programming transputers [8, 9].

Occam is a high level language that allows access to several of the special features of the transputer to optimize performance [10]. The technique of sequential loop optimization greatly reduces array access times by replacing loops with in-line code. Even though code size increases, this type of optimization is very efficient when performing arithmetic operations on vectors due to special features of the transputer instruction set. The optimization of type conversions is especially applicable to image compression since vector elements (pixels) are stored as bytes and must be converted to integers to perform distortion measure calculations. A considerable time savings can be achieved by storing the codebook bytes as integers. The amount of memory required to store the codebook quadruples with this scheme making it impractical in cases where memory is scarce.

Maximum throughput is achieved in a multi-transputer system by keeping each CPU and all links as busy as possible. This is done by the technique of overlapping communication and computation. Once a link communication operation has been initialized, data transfer can occur without significantly degrading processor performance. Use of this method triples the amount of memory required for data buffering and increases code size.

5. IMPLEMENTATION OF VECTOR QUANTIZATION ALGORITHMS

The Vector Quantization algorithms implemented in this work have been designed to achieve a compression ratio of 16. Input vectors are formed from 4x4 blocks of pixels taken from the frame grabber. Codebooks contain 256 vectors requiring 8-bit indices. Programs that perform VQ encoding, decoding, and codebook generation have been implemented for both the full search and binary tree-search algorithms. Input images are 512x512 frames of 8-bit pixels. All of the programs described here run with one VP board in the VATA. Parallelization is accomplished by distributing the codebook among the VP transputers.

5.1 Full Search Vector Quantization

The full search encoding algorithm is extremely computationally intensive requiring 200 million integer arithmetic operations to encode a single frame. Each integer arithmetic operation typically requires several instructions to complete when memory I/O and type conversions are included. Since the thirty-two T800s on the VP board have a total instruction execution rate of 320 MIPS, the computation time to encode a single frame will be several seconds. Decoding requires only a table look-up operation involving a few memory I/O instructions and takes much less time than encoding.

The VATA has been designed for real-time processing so the communication (link I/O) time is typically on the order of 33 msec. This is much less than the computation time of several seconds expected for full search VQ encoding. When the computation time is much greater than the communication time, variations in array architecture usually have little effect on overall performance. To verify this claim, the full search algorithm has been mapped onto a linear array and onto an 8-column array. Array architectures are shown in Figure 3. VI T800s are used for data transfer and reorganization only.

Encoding times for the full search algorithm are summarized in Table 1. The test image is a face with simple background. Effects of sequential loop optimization, type conversion optimization, and communication overlap have been measured for both array architectures. The type conversion optimization could not be tested in the 8-column configuration due to the 4 Kbyte memory limitation in the VP T800s. Data transfer and reorganization time is measured by removing the encoding process from the overlapped code.

Overlap of communication and computation produces less speedup in the 8-column implementation than it does in the linear implementation because the speedup from this optimization is proportional to the number of transputers that the data must pass through. The difference between data transfer and reorganization times for the two array architectures is mostly due to the fact that twice as many transputers are performing these tasks in the 8-column case. Even if this effect is neglected, the relative difference between the measured times of 5.1 and 4.8 seconds for the overlapped loop optimized cases is less than 6%. This supports the claim that performance is not strongly influenced by array architecture when computation time is dominant.

5.2 Binary Tree-Search Vector Quantization

Full search encoding requires 256 distortion measure calculations for each input vector while binary tree-search encoding requires only 8. This indicates that binary tree-search encoding time will be approximately eight times faster than full search encoding time. As a result, computation time will not be much larger than communication time and a multiple column array should perform significantly better than a linear array.

Unfortunately, the structure of the binary tree-search algorithm and the storage requirements for a larger codebook combine to make a multiple column implementation impractical due to the 4 Kbyte memory constraint in the VP T800s. Even with a linear architecture, the memory limitation prevents implementation of an optimal distribution of the codebook making encoding time image dependent. As a result, encoding time for this algorithm is expected to be between 2 and 8 times faster than encoding time for the full search algorithm. The memory limit also prevents the use of type conversion optimizations.

To measure encoding time variations, timing tests have been made using three input images. A best case image has been constructed from optimally ordered vectors taken from the last level of a default binary tree codebook. A second image in which all pixels have been set to the same value is used for a worst case test. The last image is the test image used for full search VQ.

Encoding times for the optimal, typical, and worst case input images are listed in Table 2. The encoding time for the best case image is approximately equal to the data transfer and reorganization time when optimization and overlap are included. This indicates that encoding time is no greater than 0.5 seconds, about eight times faster than the linear implementation of the full search algorithm. Data

transfer and reorganization time is halved when all four VI T800s are used in the 8-column implementation of the full search algorithm. This indicates that performance would be significantly improved by a multiple column architecture.

A "near real-time" demonstration of the binary tree VQ algorithm has been developed to simulate videophone applications. This program uses the linear array implementation of the binary tree algorithm to encode and decode 128x128 pixel images. Images are grabbed from the video camera, processed, and displayed on the monitor. This program runs at 7.5 frames/second.

Comparison with the results in Table 2 indicate that the real-time program should run about twice this fast. The limitation in this case is due to the I/O port of the frame grabber. Even with all processing removed, the maximum frame rate is 7.5 frames/second. Additional tests show that the 128x128 pixel program would run at about 12 frames/second without frame grabber limitations. Without this limitation, addition of a second VP board and inclusion of the unused VI T800s in a "two-column" architecture would more than double the frame rate for a "real-time" system. The use of 30Mhz T800s would further increase speed.

6. CONCLUSION

Computationally intensive vector quantization algorithms have been mapped onto the VATA and used to compress image data. Different array architectures have been implemented and algorithm performance has been compared for each architecture. Array architecture has little effect on performance when computation time is much greater than communication time. In these cases, memory can be very effectively traded for performance. Performance is maximized in all cases when sequential optimizations are combined with the overlap of communication and computation. Algorithms such as multi-stage VQ combine the speed of tree-search with the low memory requirements of full search. Multiple-column implementations of this type of VQ algorithm should increase performance significantly.

REFERENCES

1. Jain, A.K. "Image Data Compression: A Review," IEEE Proceedings 69, (1981): 349-389.

2. Gray, R.M. "Vector Quantization." IEEE Acoustics, Speech and Signal Processing Magazine, Apr. 1984: 4-29.

3. AT&T Inc. "AT&T DSP32 Product Family Featuring the New DSP32C." High Performance Systems, Sept. 1989: 192.

4. "i860 64-Bit Microprocessor Advance Information." Intel Corp., Apr. 1989.

5. Symanski, J.J., Bromley, K., and Henderson, T.B. "The Video Analysis Transputer Array (VATA)." Proceedings of the SPIE 32nd Annual International Technical Symposium on Optical and Optoelectronic Applied Science and Engineering, 977-34, (1988).

6. Henderson, T.B., Symanski, J.J., and Bromley, K. "Software Development on the Video Analysis Transputer Array." Proceedings of the First Conference of the North American Transputer Users Group, (1989): 117.

7. Transputer Databook. Bristol UK: Inmos Ltd., 1989: 43-112.

8. Hoare, C.A.R. Communicating Sequential Processes. London: Prentice-Hall International, 1985.

9. Hoare, C.A.R. Occam 2 Reference Manual. London: Prentice-Hall International, 1988.

10. Atkin, P. "Performance Maximization." Inmos Ltd., Mar. 1988.

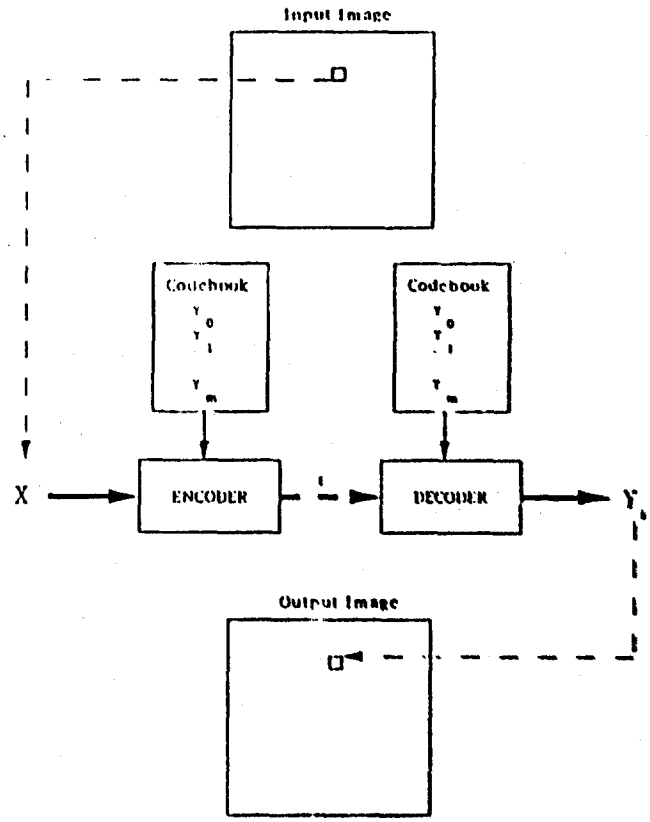


Figure 1. VQ block diagram

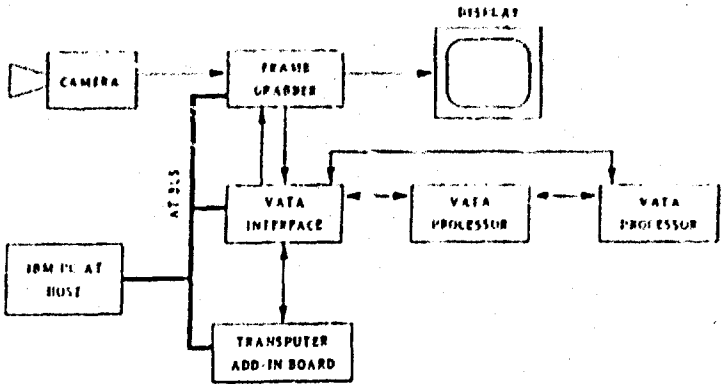
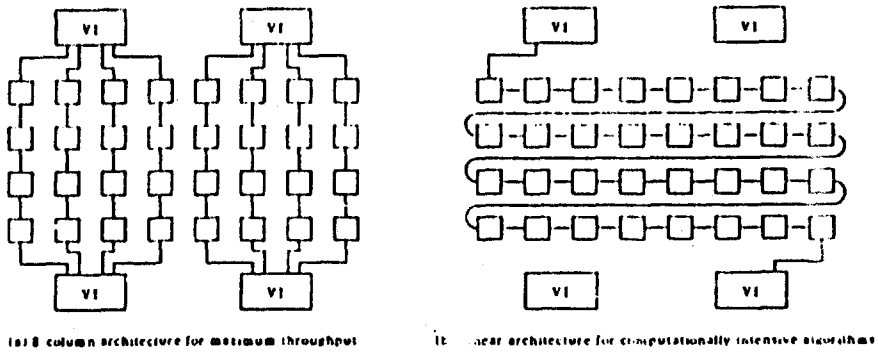


Figure 2 VATA system block diagram



(a) 8 column architecture for maximum throughput (b) Linear architecture for computationally intensive algorithms

Figure 3. Examples of VATA array architectures

Table 1. Encoding times for full search VQ

CODE OPTIMIZATIONS	ENCODING TIME	
	LINEAR ARRAY	8 COLUMN ARRAY
No modifications	8.4	7.7
Sequential loop optimizations only	5.6	4.9
Sequential loop and type conversion optimizations	4.7	***
Communication overlap only	2.7	2.6
Communication overlap with sequential loop optimizations	1.1	1.0
Communication overlap with sequential loop and type conversion optimizations	1.1	***
All processing removed to measure data transfer and reorganization time	0.8	0.4

Encoding times are in seconds per frame. Accuracy is +/- 0.8% test rate

*** These measurements could not be made due to memory limitations in the VP (Sun)

Table 2. Encoding times for binary tree search VQ

CODE OPTIMIZATIONS	ENCODING TIME		
	TEST IMAGE	WORST CASE	BEST CASE
No modifications	2.3	2.8	1.9
Sequential loop optimizations only	1.7	2.0	1.2
Communication overlap only	1.4	2.0	0.9
Communication overlap with sequential loop optimizations	0.9	1.5	0.5
All processing removed to measure data transfer and reorganization time	0.5	0.5	0.5

Encoding times are in seconds per frame. Accuracy is +/- 0.8% test rate